Permissive strategies: from parity games to safety games

Julien Bernet, David Janin, Igor Walukiewicz
LaBRI - CNRS - Université de Bordeaux I
351, cours de la Libération, F - 33405 Talence cedex
FRANCE

Abstract

It is proposed to compare strategies in a parity game by comparing the sets of behaviours they allow. For such a game, there may be no winning strategy that encompasses all the behaviours of all winning strategies. It is shown, however, that there always exists a permissive strategy that encompasses all the behaviours of all memoryless strategies. An algorithm for finding such a permissive strategy is presented. Its complexity matches currently known upper bounds for the simpler problem of finding the set of winning positions in a parity game. The algorithm can be seen as a reduction of a parity to a safety game and computation of the set of winning positions in the resulting game.

1 Introduction

An interaction of a controller and an environment can be seen as a game between two players. A correct controller is a winning strategy in such a game (see [1]). There may be many winning strategies in a given game. Often one looks for a most permissive strategy, i.e., the one that restricts the behaviour of the environment as little as possible. In general such a most permissive strategy may not exist. In this paper we propose and study a notion of permissive strategy, which is intended to capture the idea that a strategy allows "sufficiently many" behaviours.

In this paper we concentrate on parity games. These are two-player infinite games with perfect information. A game is played on a finite graph with a natural number assigned to each vertex. A move of a player is to prolong a path constructed so far. The result of an infinite play is an infinite path through the graph. A play is winning for player 0 if the smallest number among those labelling infinitely many vertices in this infinite path is even. Finding a winning strategy in a given parity game is a prominent problem in computer aided verification as many model checking problems can be effectively reduced to

it [6, 12]. Here, we are interested in the synthesis problem when the interaction of a controller and an environment is described by a parity game [17, 1].

In the context of synthesis, the advantage of considering parity games is that in a finite parity game there is a finite winning strategy whenever there is a winning strategy at all. In other words, if there is a controller then there is a finite controller. This follows from the results of Büchi [3] and Gurevich and Harrington [9] who showed that whenever there is a strategy in a game with regular winning conditions then there is a strategy with finite memory. For parity games, that are a special case of games with regular conditions, even a stronger result holds [7, 14]. It says that no memory is needed. So if there is a winning strategy in a parity game then there is a *memoryless* winning strategy that is just a subgraph of the game graph.

When considering program synthesis, correctness and size of the program is not the only criterion. For example in the theory of discrete controller synthesis [15, 2, 4] one is usually interested in the most permissive winning strategy, i.e., the one that allows most behaviours. In case of games with infinitary conditions there may be no most permissive winning strategy. Actually, as we show here, under suitable definitions most permissive strategies exist only in safety games for player 0. In such games player 0 wins if he manages to avoid a designated set of bad positions. This is one of the reasons why discrete controller synthesis theory concentrates on safety games.

Still, the setting of safety games may not be sufficient for some control synthesis problems. In fact, infinitary conditions such as fairness or liveness cannot be expressed in the context of safety games. This justifies the extension of the setting to parity games. This class of games is sufficiently expressive, as every game with regular conditions can be encoded as a parity game [13, 18].

Even though there may be no most permissive strategy in a parity game, we think that it is reasonable to compare strategies by looking at the set of behaviours they allow. A strategy permitting more behaviours is better because it is less prone to transient errors (i.e. errors that do not permit a controller to make some choices for a limited period of time). Imagine that we are in such an error situation. A more permissive strategy instead of waiting until the error will be resolved may be able to take a different action. More permissive strategy is also good for modular design. Imagine that later we are going to refine computed strategy (controller). If a strategy is too restrictive then it may not allow some admissible behaviours, and hence may not allow the required refinement.

In this paper we propose a notion of a *permissive strategy*. A strategy for a given game is permissive if it allows all the behaviours of all memoryless winning strategies in the game. We show that for every game there is a permissive strategy with finite memory. This strategy can be computed in $\mathcal{O}(n^{d/2+1})$ time and $\mathcal{O}(nd\log(n))$ space, where n is the number of vertices in the game and d is the number of different integers that are labelling vertices of the game. This matches known upper-bound for the simpler problem of computing a set of positions from which the given player has a winning strategy. The algorithm actually

turns out to be the same as strategy improvement algorithm of Jurdzinski [11].

To work with permissive strategies we introduce a reduction from parity to safety games. We find this reduction very useful. It implies that to solve parity games, it is enough to know how to solve safety games. Of course the resulting safety game may be exponentially bigger, but then the problem reduces to clever exploration of the search space. For example, the algorithm mentioned in the previous paragraph needs only linear space in the size of the initial parity game. This reduction may be useful when considering distributed or on-the-fly algorithms for game solving or model-checking [8].

The plan of the paper is as follows. In the preliminaries section we introduce parity games and strategies. We also define a permissiveness ordering on strategies and the notion of permissive strategy. In section 3 we show that only safety games have a maximally permissive strategy. Section 4 shows how to reduce the problem of finding a permissive strategy in a game to the problem of finding a maximally permissive strategy in a safety game. In the next section we discuss some additional properties of our permissive strategies. These are used in Section 6 to show correctness of the algorithm finding a permissive strategy. In the conclusions section we discuss some open problems.

2 Preliminaries

In this section we introduce parity games, winning strategies and permissive strategies. Our definitions will allow nondeterministic strategies, i.e., the strategies that may permit more than one move from a given position.

Definition 1 (Parity game) A *parity game* is a labelled graph

$$G = \langle V, V_0, V_1, E \subseteq V \times V, I, \Omega : V \to I \rangle$$

with the partition $V_0, V_1 \subseteq V$ of the set of vertices V. Additionally, I is a finite set of natural numbers, called *priorities* and Ω is a function assigning a priority to each vertex. We say that a vertex v' is a *successor* of a vertex v if $E(v, v')$ holds.

A play from some vertex $v \in V$ proceeds as follows: if $v \in V_0$ then player 0 chooses a successor of v otherwise it is player 1 who makes the choice. Then the play enters the newly chosen vertex v' and the process repeats. It continues like this ad infinitum unless one of the players cannot make a move (i.e. he has to make a choice and the vertex has no successors). If a player cannot make a move he looses. The result of an infinite play is an infinite path $vv_1v_2\ldots$ This path is *winning* for player 0 if it satisfies the *parity condition* which is:

$$\liminf_{i\to\infty} \Omega(v_i) \quad \text{is even.}$$

In other words, the condition says that the smallest among priorities appearing infinitely often in the sequence should be even.

A *strategy* σ for player 0 is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex v from V_0 a set $\sigma(\vec{v}) \subseteq V$ of successors of v. We require that $\sigma(\vec{v}) \neq \emptyset$ if v has a successor. Intuitively a strategy depends on the sequence $\vec{v}$ of moves made so far, moreover it should allow some move if a move is possible. A *play respecting strategy* σ is a finite or infinite path $v_0 v_1 \dots$ such that $v_{i+1} \in \sigma(v_0 \dots v_i)$ for every i with $v_i \in V_0$. A *maximal play* is an infinite path or a finite path ending in a vertex with no successors. So, a maximal play is winning for player 0 if it satisfies the parity condition or ends in a vertex from V_1.

Definition 2 A strategy for player 0 *is winning from a vertex* v iff every maximal play starting in v and respecting the strategy is winning for player 0. We say that a strategy for player 0 is *winning* if it is winning from every vertex from which there is a winning strategy for player 0.

A *strategy with memory* M is a triple:

$$c : M \times V_0 \to \mathcal{P}(V), \quad up : M \times V \to M, \quad m_0 \in M$$

The role of the initial memory element m_0 and the memory update function up is to abstract some information from the sequence $\vec{v}$. This is done by iteratively applying up function:

$$up^*(m, \varepsilon) = m \quad \text{and} \quad up^*(m, \vec{v}v) = up^*(up(m, \vec{v}), v)$$

This way each sequence $\vec{v}$ of vertices is assigned a memory element $up^*(m_0, \vec{v})$. Then the choice function c defines a strategy by $\sigma(\vec{v}v) = c(up^*(m_0, \vec{v}), v)$. When up and m_0 will be clear from the context, we will sometimes use σ to denote the function c.

A *memoryless strategy* is a strategy with memory M which is a singleton set. Alternatively one can see it as a function $\sigma : V \to \mathcal{P}(V)$.

Comparing strategies The following are the main definitions of the paper. We define the set of behaviours allowed by a strategy. Then we say that a strategy subsumes another strategy if it allows more behaviours. Finally, we say that a strategy is permissive if it subsumes all memoryless strategies.

Definition 3 If σ is a strategy and v is a vertex of G from which σ is winning then $Beh(G, v, \sigma)$ is the set of all plays starting in v and respecting σ. If σ is not winning from v then we put $Beh(G, v, \sigma) = \emptyset$.

The intuition behind this definition is that we are only interested in the part of σ which guarantees win for player 0. Note that $Beh(G, v, \sigma)$ is prefix closed.

Lemma 4 All maximal paths in $Beh(G, v, \sigma)$ are winning for player 0. If all prefixes of some infinite path belong to $Beh(G, v, \sigma)$ then the path belongs to $Beh(G, v, \sigma)$.

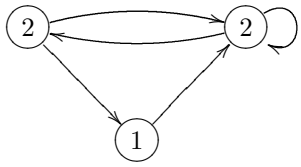
Proof
The first statement follows directly from the definition. For the second statement observe that if all the prefixes of a path are allowed by σ then the whole path is allowed by σ. □

We can compare the strategies by comparing the behaviours that they allow.

Definition 5 A strategy σ' is *subsumed* by σ, which is denoted $\sigma' \sqsubseteq \sigma$, if $Beh(G, v, \sigma') \subseteq Beh(G, v, \sigma)$ for all $v \in G$.

Definition 6 A strategy σ is *permissive* if $\sigma' \sqsubseteq \sigma$ for every memoryless strategy σ'.

Example: Here is a game that has two $\sqsubseteq$-maximal memoryless strategies. All the positions are for player 0. The priorities of vertices are as indicated.



The first maximal memoryless strategy allows all but the edge from the leftmost vertex to 1. The second strategy allows all but the edge from the rightmost to the leftmost vertex.

Any permissive strategy for this game needs a memory. In the the rightmost vertex the strategy should allow to go to the leftmost vertex only if we have not passed through 1 at any time before. There are other permissive strategies. For example, a strategy can allow to go from the rightmost to the leftmost vertex provided we have passed through 1 less then, say, 5 times. This is intuitively the reason why there does not exist the most permissive strategy. □

Remark: One may ask why to limit oneself to permissive strategies. It would be the best just to find the $\sqsubseteq$-biggest strategy. Unfortunately, as explained in the next section, such strategies exist only for very simple games.

Remark: One can define M-permissive strategies, which would be the strategies subsuming all strategies with memory of size M. The approach presented here extends to this setting, but we have chosen not to consider such strategies due to a substantial notational overhead.

3 Safety games

A safety game is a special kind of parity game where player 0 wins a play if it never enters any of forbidden positions. More formally, a safety game is a game

$$G = \langle V, V_0, V_1, E \subseteq V \times V, I, \Omega : V \to I \rangle$$

with $I = \{0, 1\}$ and the property that for every vertex v of priority 1:

- if $v \in V_0$ then all successors of v must have priority 1, and
- if $v \in V_1$ then there must exist a successor of v with priority 1.

The definition may at first seem too complicated, but we want it to be general and we also need to forbid situations like having a vertex for player 1 with no successors. According to our definition, such a vertex would be winning for player 0.

Fact 7 In a safety game player 0 has a $\sqsubseteq$-biggest winning strategy. This strategy is memoryless.

Proof
To calculate the set W of winning positions for player 0 in a game G, one can proceed as follows. First, one sets $W = \{v : \Omega(v) = 0\}$. Then, repeatedly one tries to find a vertex v such that either:

- $v \in V_0$ and all successors of v are not in W, or
- $v \in V_1$ and there is a successor of v not in W.

One removes v from W and repeats the process. The loop ends when there are no more vertices to remove. It is not difficult to show that from every removed vertex player 1 can force the play to a vertex of priority 1. On the other hand player 0 has a strategy to stay in W. This strategy is given by $\sigma(v) = \{v' \in W : E(v, v')\}$. The strategy is maximal as no winning strategy can allow a play to get outside W. □

The following fact shows that if there is a $\sqsubseteq$-biggest winning strategy σ in a game, then the game is essentially a safety game, and σ is the "stay in the winning set" strategy.

Fact 8 If a game G has a $\sqsubseteq$-biggest winning strategy σ, then one can assign to each vertex of G a priority in $\{0, 1\}$ in such a way that the result G' is a safety game and σ is also the $\sqsubseteq$-biggest winning strategy in G'.

Proof
Let W be the set of winning positions in G. If every path through W is winning for player 0 then we are done. We put $\Omega(v) = 0$ for all the vertices in W and $\Omega(v) = 1$ for all other vertices.

Suppose that there is a maximal path P in W which is not winning for player 0. This cannot be a finite path as every vertex in $W \cap V_0$ has a successor in W.

We show that if there is an infinite path P in W which is not winning then there is no $\sqsubseteq$-biggest winning strategy for player 0. Take some winning strategy σ. For every finite prefix $\vec{u}$ of P we are going to define a strategy $\sigma_{\vec{u}}$ that extends σ by allowing moves along $\vec{u}$. When the play goes out of $\vec{u}$, or the $\vec{u}$ finishes, the strategy becomes the same as σ. Formally:

$$\sigma_{\vec{u}}(\vec{v}v) = \begin{cases} \{v'\} & \text{if } \vec{v}vv' \text{ is a prefix of } \vec{u} \\ \sigma(\vec{v}_1 v) & \text{if } \vec{v} = \vec{v}_0\vec{v}_1 \text{ and } \vec{v}_0 \text{ is the longest common prefix} \\ & \text{of } \vec{v} \text{ and } \vec{u}. \end{cases}$$

Every play respecting $\sigma_{\vec{u}}$ has a suffix which is a play starting from some $v \in W$ and respecting σ. Hence, every play respecting $\sigma_{\vec{u}}$ is winning.

Suppose that we have τ such that $Beh(G, v, \sigma_{\vec{u}}) \subseteq Beh(G, v, \tau)$ for all prefixes $\vec{u}$ of P. Then we have $P \in Beh(G, v, \tau)$ as all its prefixes are in the set. But this is impossible as P is not winning. □

Remark: The above fact does not hold if we only require that there is a $\sqsubseteq$-biggest among memoryless strategies in G. A simple example is the following game where all vertices are for player 0 and the priorities are as indicated.

In this game player 0 wins from both vertices and there is a unique memoryless strategy. Still there is a path that is loosing for player 0. So "stay in the winning set" strategy is not winning for him.

Remark: In the above fact we consider relabelings that do not change the biggest strategy. In other words, the biggest strategies in the original and the relabeled game are the same. Let us explain why we have not considered the relabelings that only preserve the sets of winning vertices. According to this weaker requirement every game can be relabeled to a safety game. One just puts $\Omega(v) = 0$ for all the vertices winning for player 0 and $\Omega(v) = 1$ for all the vertices winning for player 1. After this relabeling the sets of winning vertices do not change as player 0 has a strategy to stay in his winning vertices, and player 1 has a strategy to stay in his winning vertices. For example, consider the game from the remark above. If one changes the priorities of both vertices to 0 then one gets a game with the same set of winning positions but with a new $\sqsubseteq$-biggest winning strategy. This strategy allows a path going infinitely often through vertex 1. This path was loosing in the original game.

4 Finding permissive strategies

In this section we will show that there are finite memory permissive strategies. It can be shown that there cannot be a memory size that is sufficient for a permissive strategy in every game. Still we can hope to have one uniform memory for all the games of fixed size. There is a similar situation in the case of games with Muller conditions [18, 20, 5]. There, the size of memory also cannot in general be bounded, but there is a finite memory sufficient for all games with conditions over some fixed set of elements.

For the rest of this section let us fix a set $I = \{0, \ldots, d+1\}$ of priorities and a number n_p for each odd priority $p \in I$. For convenience let us assume that d is odd. Let $\vec{n} = (n_1 n_3 \ldots n_d)$. This vector will be used to bound the size of considered games.

Definition 9 A game is $\vec{n}$ *bounded* if its vertices have priorities from $\{0, \ldots, d+1\}$ and there are at most n_p vertices of priority p, for each odd p.

In this section we will show a uniform construction of permissive strategies in $\vec{n}$-bounded games. For this we define a memory set $M(\vec{n})$ that will be used by our strategies.

$$M(\vec{n}) = \prod_{1 \leq p \leq d,\ p \text{ odd}} \{0, \ldots, n_p\}$$

An element $\vec{m} \in M(\vec{n})$ is a tuple of numbers $(m_1, m_3, \ldots, m_d)$ with $0 \leq m_i \leq n_i$. We can consider such a tuple as a counter representing the number

$$\sum_{i=1,3,\ldots,d} m_i \Big(\prod_{j=i+2,i+4,\ldots,d} (n_j + 1) \Big)$$

So the most significant digit is the first one and each position p is in base n_p. For example, in the simple case when $n_p = 1$ for all p, we get a binary encoding of numbers up to $2^{(d+1)/2} - 1$.

The plan for finding a permissive strategy is the following. First, we will take $M_\top(\vec{n})$ which is an extension of $M(\vec{n})$ with an element $\top$ standing for overflow. Then, we will define a uniform memory update function $up : M_\top(\vec{n}) \times I \to M_\top(\vec{n})$. We call it uniform because it does not depend on vertices of a particular game but only on the priorities (and these are the same for all the games in question). Memory $M_\top(\vec{n})$ will allow to reduce a game G to a safety game $G^\otimes$. The biggest strategy in this game will in turn be used to get a permissive strategy in G.

To define the memory update function we need to define two kinds of auxiliary functions on memories: $\vec{m}|_p$ and $inc_p(\vec{m})$ for every $p \in I$. The first is just resetting to 0 all the positions bigger than p. For odd p it is defined by:

$$(m_1, m_3, \ldots, m_d)|_p = (m_1, \ldots, m_p, 0, \ldots, 0)$$

and for even p it is the same as $\vec{m}|_{p+1}$.

The other operation is like adding 1 to position p when considering $\vec{m}$ as a counter; if the value on this position is already n_p then we try recursively to add 1 to the position $p - 2$:

$$inc_p((m_1, \ldots, m_d)) = \begin{cases} (m_1, \ldots, m_p + 1, 0, \ldots, 0) & \text{if } m_p < n_p \\ inc_{p-2}((m_1, \ldots, m_d)) & \text{if } m_p = n_p \text{ and } p \geq 3 \\ \top & \text{otherwise} \end{cases}$$

The intuition for the last case of the above definition is that if the value of the counter on first p positions is $n_1 n_2 \ldots n_p$ then adding 1 is impossible and the value is $\top$ which denotes an overflow.

Now, we can define a generic update function $up : M_\top(\vec{n}) \times I \to M_\top(\vec{n})$. For clarity, from now on we will often not write vectors over m.

$$up(m, p) = \begin{cases} m|_p & \text{for } p \text{ even} \\ inc_p(m) & \text{for } p \text{ odd} \end{cases}$$

Of course we also have $up(\top, p) = \top$ which means that there is no possibility to recover from the overflow.

Using the memory $M_\top(\vec{n})$ and the function up we can reduce any $\vec{n}$ bounded game G to a safety game. Let us take an $\vec{n}$-bounded game $G = \langle V, V_0, V_1, E, I, \Omega\rangle$. Define a safety game $G^\otimes = \langle V^\otimes, V_0^\otimes, V_1^\otimes, E^\otimes, \{0,1\}, \Omega^\otimes\rangle$, where:

- $V_i^\otimes = V_i \times M_\top(\vec{n})$, for $i = 0, 1$;
- $E^\otimes((v,m),(v',m'))$ if $E(v,v')$ and $m' = up(m, \Omega(v))$;
- $\Omega^\otimes((v,m)) = 0$ if $m \neq \top$ and $\Omega^\otimes((v,\top)) = 1$.

So player 0 wins in $G^\otimes$ from a position (v,m) if he has a strategy to avoid vertices with $\top$ in the second component. By Fact 7, in such a game there is always a maximal memoryless winning strategy.

A memoryless strategy $\sigma^\otimes$ in $G^\otimes$ gives a strategy σ with memory $M(\vec{n})$ in G. The strategy is defined by $\sigma(m,v) = \sigma^\otimes((v,m))$, the initial memory element is $m_0 = (0,\ldots,0)$ and the memory update function is $up(m,v) = up(m, \Omega(v))$.

Lemma 10 For every $\vec{n}$ bounded game G; if $\sigma^\otimes$ is a memoryless strategy winning from (v,m) in $G^\otimes$ then σ is a winning strategy from v with initial memory m.

Proof
The main observation is that if we have an infinite play $(v_1,m_1)(v_2,m_2)\ldots$ and $\top$ does not appear in the sequence, then the sequence $v_1v_2\ldots$ satisfies the parity condition. Suppose the contrary; then some odd priority p would be the smallest one appearing infinitely often in $v_1v_2\ldots$ But then, by the definition of up function, we will get $\top$ after meeting $(n_1 \cdot n_3 \cdots n_p + 1)$ times a vertex of priority p and not meeting any vertex of smaller priority in between.

To see that σ is winning from v with initial memory m it is enough to note that for every play $vv_1v_2\ldots$ from v respecting σ there is a sequence of memories $mm_1m_2\ldots$ such that $(v,m)(v_1,m_1)(v_2,m_2)\ldots$ is a play from (v,m) respecting $\sigma^\otimes$. □

There is also a construction in the opposite direction. A memoryless strategy τ in G defines a memoryless strategy $\tau^\otimes$ in $G^\otimes$ by:

$$\tau^\otimes(v,m) = \{(v', up(m, \Omega(v))) : v' \in \tau(v)\}$$

Lemma 11 For every $\vec{n}$ bounded game G and every memoryless strategy τ for player 0. If τ is a winning strategy from v then $\tau^\otimes$ is winning from $(v,(0,\ldots,0))$ in $G^\otimes$.

Proof
Suppose that $\tau^\otimes$ is not winning from (v,m_0) where $\vec{m}_0 = (0,\ldots,0)$. Then there is a finite path $(v,\vec{m}_0)(v_1,\vec{m}_1)(v_2,\vec{m}_2)\ldots(v_{k+1},\vec{m}_{k+1})$ such that $\vec{m}_{k+1} = \top$. This can happen only because $\vec{m}_k = (n_1, n_3, \ldots, n_q, \ldots)$ and $\Omega(v_k) = q$, i.e., the counter $\vec{m}_{k+1}$ overflows.

Let i be the smallest integer such that $m_{i,p} = n_p$, where $p = \Omega(v_i)$ and $\vec{m}_i = (m_{i,1}, m_{i,3}, \dots)$. So we take the first vertex where the counter reaches the maximal value on the position corresponding to the priority of the vertex. Unlike in the paragraph above we do not require that all smaller positions have maximal values. So p may be different from q. Take the largest $j < i$ s.t. $\Omega(v_j)$ is both even and less than p (or take $j = -1$ if there is no such vertex). By definition of *up* function we have $m_{j+1,p} = 0$. By the choice of i, in all memories up to i no position reaches its maximal allowed value. So by the definition of *up* function, the value on position p can increase only when we see a vertex of priority p. Hence, there must exist $n_p + 1$ occurrences of vertices of priority p between v_j and v_i. As game G is $\vec{n}$ bounded, some vertex must occur twice. This is a contradiction with the fact that $vv_1v_2 \dots v_k$ is a play respecting τ. On such a play there cannot be a loop through a vertex of odd priority p without a vertex of smaller priority on this loop since τ is winning. □

Theorem 12
For a given $\vec{n} = (n_1, n_3, \dots, n_d)$. For every $\vec{n}$-bounded game G there is a permissive strategy on G using memory $M_\top(\vec{n})$.

Proof
Let $\sigma^\otimes$ be the maximal winning strategy in the game $G^\otimes$. This defines in G a strategy σ with memory $M_\top(\vec{n})$. The strategy is winning by Lemma 11. We want to show that it is a permissive strategy. For this we take some memoryless winning strategy τ in G and show that $Beh(G, v_0, \tau) \subseteq Beh(G, v_0, \sigma)$ for every v_0.

Take $v_0v_1 \dots \in Beh(G, v, \tau)$. By Lemma 11, there are memories such that $(v_0, m_0)(v_1, m_1) \dots \in Beh(G^\otimes, (v, m), \tau^\otimes)$. Next, by the maximality of $\sigma^\otimes$, we know that $Beh(G^\otimes, (v, m), \tau^\otimes) \subseteq Beh(G^\otimes, (v, m), \sigma^\otimes)$ for every (v, m). Hence, $(v_0, m_0)(v_1, m_1) \dots \in Beh(G^\otimes, (v_0, m_0), \sigma^\otimes)$. Finally, by the definition of σ we have that $v_1v_2 \dots \in Beh(G, v, \sigma)$ □

Remark: The memory as defined above is essentially nothing more than a deterministic automaton accepting sequences satisfying a parity condition. The important point is that this automaton is a safety automaton. It is well known that deterministic safety automata cannot recognize the language of all the sequences satisfying a parity condition [16]. We overcome this problem by limiting the number of odd priorities that can appear in the sequence without a smaller even priority in between. Some other solutions are also possible giving some other memories and some other permissive strategies.

5 Small representations of permissive strategies

In the previous section we have seen that for every game G there is a permissive strategy that can be represented as the biggest strategy in $G^\otimes$. The size of $G^\otimes$ is $(|G| \cdot n_1 \cdot n_3 \cdots n_d)$, hence it is exponential in the size of G. So at first glance it may seem that we need this much space to describe a permissive strategy.

Fortunately it is not the case. Here we will show that a permissive strategy can be determined by a function $Mmax : V \to M(\vec{n})$, i.e., a function assigning one memory value to each node.

The key observation is that the lexicographic ordering on memories is also a "permissiveness" ordering. We say that $\vec{m}' = (m'_1, m'_3, \ldots, m'_d)$ is *lexicographically smaller* than $\vec{m} = (m_1, m_3, \ldots, m_d)$, denoted $\vec{m}' <_L \vec{m}$, if there is a p such that $m'_p \neq m_p$, and $m'_p < m_p$ for the smallest such p. We extend this ordering by two new elements $\bot$ and $\top$ with $\bot <_L \vec{m} <_L \top$ for every $\vec{m} \in M(\vec{n})$. These two elements signify undefined and overflow respectively. Element $\top$ was already introduced in the previous section.

Lemma 13 For every game $G^{\otimes}$: if player 0 has a winning strategy from a position $(v, \vec{m})$ then he has a winning strategy from position $(v, \vec{m}')$ for every $\vec{m}' <_L \vec{m}$.

Proof
For the proof it is enough to observe that *up* function is monotonic, i.e., for every priority p: $up(\vec{m}', p) \leq_L up(\vec{m}, p)$ if $\vec{m}' \leq_L \vec{m}$. In particular for overflow it means that: if $up(\vec{m}', p) = \top$ and $\vec{m}' <_L \vec{m}$ then $up(\vec{m}, p) = \top$. □

For each vertex v, let $Mmax(v)$ be the $<_L$-supremum of all the memories m such that (v, m) is winning for player 0 in $G^{\otimes}$. So, if there is no such memory then $Mmax(v) = \bot$. By Lemma 11, $Mmax(v) = \bot$ iff v is not winning for player 0 in G. By definition, $Mmax(v)$ can never be $\top$.

We can use $Mmax(v)$ to get a permissive strategy. It is defined by telling for every v for which memories m the position (v, m) is winning in $G^{\otimes}$. As $Mmax(v)$ gives the biggest such m, we know that (v, m) is winning for exactly those m that are lexicographically not bigger than $Mmax(v)$. So in a vertex v with memory $m \leq_L Mmax(v)$ the strategy is $\sigma(m, v) = \{v' : up(m, \Omega(v)) \leq_L Mmax(v')\}$.

6 Algorithmic issues

Here we will describe how to use the reduction from G to $G^{\otimes}$ in algorithms for solving parity games, i.e., algorithms that find the set of vertices from which player 0 has a winning strategy.

A simple algorithm for solving a $\vec{n}$ bounded game G is to construct $G^{\otimes}$ and solve this safety game. This can be done by any alternating reachability algorithm. The size of $G^{\otimes}$ is $(|G| \cdot n_1 \cdot n_3 \cdots n_d)$, where n_p is the number of vertices of priority p in G. Hence, the time complexity of this algorithm is as good as the best known upper bounds for solving parity games. The weakness of this approach, however, is that a memory needed for alternating reachability algorithm is proportional to the size of the game, and hence exponential in the number of priorities.

Yet, a better approach is available. The idea is to calculate *Mmax* function in a bottom-up way. Before presenting the algorithm we need to define a function

down. For a memory m and a priority p, we put

$$down(m,p) = \max\{m' : up(m',p) \leq m\}$$

Hence, the value of $down(m,p)$ can be $\perp$ if $m = (0,\ldots,0)$. It is easy to check that $down(m,p)$ can be defined in a similar way to $up(m,p)$:

$$down(m,p) = \begin{cases} m|^p & \text{if } p \text{ even} \\ dec_p(m) & \text{if } p \text{ odd} \end{cases}$$

where for every even p and every odd q we have:

$$(m_1,\ldots,m_d)|^p = (m_1,\ldots,m_{p-1},n_{p+1},\ldots,n_d)$$

$$dec_q(m_1,\ldots,m_d) = \begin{cases} (m_1,\ldots,m_q-1,n_{q+1},\ldots,n_d) & \text{if } m_q > 0 \\ dec_{q-2}(m_1,\ldots,m_d) & \text{if } m_q = 0 \text{ and } q \geq 3 \\ \perp & \text{otherwise} \end{cases}$$

The algorithm calculating function *Mmax* will work with the auxiliary assignment $F : V \to (M(\vec{n}) \cup \{\perp\})$. Initially we put $F(v) = \vec{n}$ for each v; recall that $\vec{n} = (n_1, n_3, \ldots, n_d)$. Afterwards, we start a loop were we find a vertex v such that

$$F(v) >_L down(m', \Omega(v))$$

where

$$m' = \begin{cases} \max\{F(v') : v' \text{ successor of } v\} & \text{if } v \in V_0 \\ \min\{F(v') : v' \text{ successor of } v\} & \text{if } v \in V_1 \end{cases}$$

For such v we set $F(v) = down(m', \Omega(v))$ and repeat the loop. We stop when we cannot find a vertex with the above property. We show below that at the end $F(v) = Mmax(v)$ for all vertices v.

Remark: The algorithm is just a computation of the greatest fixpoint of some operator on $V \to (M(\vec{n}) \cup \{\perp\})$. The lemmas below make it more explicit.

Lemma 14 If $F : V \to (M(\vec{n}) \cup \{\perp\})$ is such that the value of no vertex can be decreased then $F(v) \leq_L Mmax(v)$ for all vertices v.

Proof
It is enough to show that for every v with $F(v) \neq \perp$ the position $(v, F(v))$ in $G^{\otimes}$ is winning for player 0. The observation we need is that if F is as in the assumption of the lemma then for every v s.t. $F(v) \neq \perp$ we have:

- if $v \in V_0$ then there must be a successor v' with $up(F(v), \Omega(v)) \leq_L F(v')$;
- if $v \in V_1$ then for all successors v' of v we have $up(F(v), \Omega(v)) \leq_L F(v')$.

Now the strategy for player 0 is to choose in every $v \in V_0$ a successor v' such that $up(F(v), \Omega(v)) \leq_L F(v')$. By the above this is possible for every vertex with $F(v) \neq \bot$. To see that this strategy is winning take a play $(v_1, m_1)(v_2, m_2)\ldots$ respecting the strategy where $m_1 = F(v_1) \neq \bot$. Using the property above we get by induction on i that $m_i \leq_L F(v_i)$. Hence, $m_i \neq \top$ for all i, which means that the play is winning. □

Lemma 15 After each iteration of the above loop we have $F(v) \geq_L Mmax(v)$ for all vertices v.

Proof
The proof is by induction on the number of iterations. The statement is true at the beginning when $F(v) = \vec{n}$ for all v. For the induction step we assume that $F(v) \geq_L Mmax(v)$ holds for all v and we choose one v for which $F(v)$ can be decreased.

Suppose that we have chosen $v \in V_0$ and it is to be decreased. We need to show that the new value of $F(v)$ is still not smaller than $Mmax(v)$. If $Mmax(v) = \bot$ then we are done. Otherwise, as $Mmax(v)$ is a memory that still guarantees a win for player 0, we know that v has a successor v' with $up(Mmax(v), \Omega(v)) \leq_L Mmax(v')$. Applying *down* function to both sides we get:

$$Mmax(v) \leq_L down(up(Mmax(v), \Omega(v)), \Omega(v)) \leq_L down(Mmax(v'), \Omega(v))$$

The first inequality follows by the property: $m \leq_L down(up(m, p), p)$ for every $m \in M(\vec{n})$. The second inequality follows from the monotonicity of *down*. The new value of $F(v)$ is not smaller than $down(F(v'), \Omega(v))$. So we are done as

$$down(F(v'), \Omega(v)) \geq_L down(Mmax(v'), \Omega(v)) \geq_L Mmax(v)$$

The case for $v \in V_1$ is similar. □

Corollary 16 At the end of the algorithm $F(v) = Mmax(v)$.

Let us calculate the complexity of the algorithm. It cannot do more than than $(|G| \cdot (n_1 + 1) \cdot (n_3 + 1) \cdots (n_d + 1))$ steps. This is because at each step the F value of some node is decreased and the value of a node cannot be decreased more than $(n_1 + 1) \cdot (n_3 + 1) \cdots (n_d + 1)$ times. The algorithm uses $n \log((n_1+1) \cdot (n_3+1) \cdots (n_d+1))$ memory, as it needs to store just the current values of F assignment. This matches the best known upper bounds for solving parity games [11]. The known upper bound presently known for the strategy improvement algorithm [19] is actually worse: $(n/d)^d$ instead of $(n/d)^{\lceil d/2 \rceil}$.

7 Conclusions

Learning from the experience of discrete control synthesis theory, it seems to be a good idea to compare strategies by comparing the sets of behaviours they

allow. As we presented above, there are parity games where there is no winning strategy that allows all the behaviours of all possible winning strategies in the game. Given this, we propose a more lax notion of permissive strategy which is a strategy that allows all the behaviours of all memoryless strategies. We show that a permissive strategy exists for every game and that the algorithm finding it has not worse complexity than currently known algorithms for a simpler problem of deciding if there is any winning strategy from a given vertex. Actually, the algorithm we obtain is exactly the signature improvement algorithm presented in [11]. Hence, we show that this algorithm computes more than just a set of winning vertices (and some winning strategy).

There are at least two interesting open problems. The first concerns the size of permissive strategy. We have shown that for an $\vec{n} = (n_1, \ldots, n_d)$ bounded game there is a strategy with memory of size $n_1 \cdot n_2 \cdots n_d$. We don't known whether there can be a memory of smaller size. Actually if there were a memory of size polynomial in $n_1 + n_2 + \cdots + n_d$ then it would give a PTIME algorithm for solving parity games. Our reduction to safety games shows that the question about minimal memory is equivalent to the question about automata on infinite words. The goal is to find a minimal automaton accepting all paths that are admitted by some memoryless strategy in some $\vec{n}$-bounded game.

The other problem also concerns complexity. We have shown that a permissive strategy in a game is defined by a function $Mmax : V \to (M(\vec{n}) \cup \bot)$. This function is unique for a given game. Hence, if we were able to check in PTIME that a given function $F : V \to (M(\vec{n}) \cup \bot)$ is exactly the $Mmax$ function then we would show that solving parity games is in UP∩co-UP. This would be interesting as the known arguments for UP∩co-UP bound are indirect and go through discounted payoff games [10, 21].

Acknowledgments

The authors thank André Arnold for many stimulating discussions and useful comments.

References

[1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *TCS*, 2002. to appear.

[2] A. Bergeron. A unified approach to control problems in discrete event processes. *RAIRO-ITA*, 27:555–573, 1993.

[3] J. R. Buchi. State strategies for games in $F_{\sigma\delta} \cap G_{\delta\sigma}$. *Journal of Symbolic Logic*, 48:1171–1198, 1983.

[4] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[5] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How much memory is needed to win infinite games. In *LICS*, pages 99–110, 1997.

[6] E. A. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *CAV'93*, volume 697 of *LNCS*, pages 385–396, 1993.

[7] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS 91*, pages 368–377, 1991.

[8] O. Grumberg, T. Heyman, and A. Schuster. Distributed symbolic model checking for mu-calculus. In *CAV'01*, volume 2102 of *LNCS*, 2001.

[9] Y. Gurevich and L. Harrington. Trees, automata and games. In *14th ACM Symp. on Theory of Computations*, pages 60–65, 1982.

[10] M. Jurdziński. Deciding the winner in parity games is in UP∩co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[11] M. Jurdziński. Small progress measures for solving parity games. In *STACS*, volume 1770 of *LNCS*, pages 290–301, 2000.

[12] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2), 2000.

[13] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In A. Skowron, editor, *Fifth Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.

[14] A. W. Mostowski. Hierarchies of weak automata and week monadic formulas. *Theoretical Computer Science*, 83:323–335, 1991.

[15] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77, 1989.

[16] W. Thomas. Automata on infinite objects. In J. van Leeuven, editor, *Handbook of Theoretical Computer Science Vol.B*, pages 133–192. Elsevier, 1990.

[17] W. Thomas. On the synthesis of strategies in infinite games. In *STACS '95*, volume 900 of *LNCS*, pages 1–13, 1995.

[18] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997.

[19] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, 2000.

[20] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinte trees. *Theoretical Computer Science*, 200:135–183, 1998.

[21] U. Zwick and M. Paterson. The complexity of mean payoff games on graps. *Theoretical Computer Science*, 158:343–359, 1996.